Kubernetes中的资源隔离以及控制

Kubelet ---> docker --->system中的cgroup机制

Namespace与cgroup的区别

cgroup和namespace类似，也是将进程进行分组，但它的目的和namespace不一样，namespace是为了隔离进程组之间的资源，而cgroup是为了对一组进程进行统一的资源监控和限制。

Cgroup用于统一对进程进行分组,并在分组的基础上对进程进行监控和资源控制管理.

cgroup是Linux下的一种将进程按组进行管理的机制，在用户层看来，cgroup技术就是把系统中的所有进程组织成一颗一颗独立的树，每棵 树都包含系统的所有进程，树的每个节点是一个进程组，而每颗树又和一个或者多个subsystem关联，树的作用是将进程分组，而subsystem的作 用就是对这些组进行操作。cgroup主要包括下面两部分：

**subsystem** 一个subsystem就是一个内核模块，他被关联到一颗cgroup树之后，就会在树的每个节点（进程组）上做具体的操作。subsystem经常被称 作"resource controller"，因为它主要被用来调度或者限制每个进程组的资源，但是这个说法不完全准确，因为有时我们将进程分组只是为了做一些监控，观察一下 他们的状态，比如perf\_event subsystem。到目前为止，Linux支持12种subsystem，比如限制CPU的使用时间，限制使用的内存，统计CPU的使用情况，冻结和恢 复一组进程等，后续会对它们一一进行介绍。

**hierarchy** 一个hierarchy可以理解为一棵cgroup树，树的每个节点就是一个进程组，每棵树都会与零到多个subsystem关联。在一颗树里面，会包含 Linux系统中的所有进程，但每个进程只能属于一个节点（进程组）。系统中可以有很多颗cgroup树，每棵树都和不同的subsystem关联，一个 进程可以属于多颗树，即一个进程可以属于多个进程组，只是这些进程组和不同的subsystem关联。目前Linux支持12种subsystem，如果 不考虑不与任何subsystem关联的情况（systemd就属于这种情况），Linux里面最多可以建12颗cgroup树，每棵树关联一个 subsystem，当然也可以只建一棵树，然后让这棵树关联所有的subsystem。当一颗cgroup树不和任何subsystem关联的时候，意 味着这棵树只是将进程进行分组，至于要在分组的基础上做些什么，将由应用程序自己决定，systemd就是一个这样的例子。

## 如何查看当前系统支持哪些subsystem

可以通过查看/proc/cgroups(since Linux 2.6.24)知道当前系统支持哪些subsystem，下面是一个例子

#subsys\_name hierarchy num\_cgroups enabled

cpuset 11 1 1

cpu 3 64 1

cpuacct 3 64 1

blkio 8 64 1

memory 9 104 1

devices 5 64 1

freezer 10 4 1

net\_cls 6 1 1

perf\_event 7 1 1

net\_prio 6 1 1

hugetlb 4 1 1

pids 2 68 1

从左到右，字段的含义分别是：

subsystem的名字

subsystem所关联到的cgroup树的ID，如果多个subsystem关联到同一颗cgroup树，那么他们的这个字段将一样，比如这里的cpu和cpuacct就一样，表示他们绑定到了同一颗树。如果出现下面的情况，这个字段将为0：

|  |
| --- |
| 当前subsystem没有和任何cgroup树绑定  当前subsystem已经和cgroup v2的树绑定  当前subsystem没有被内核开启 |

subsystem所关联的cgroup树中进程组的个数，也即树上节点的个数

1表示开启，0表示没有被开启(可以通过设置内核的启动参数“cgroup\_disable”来控制subsystem的开启).

## 所有的subsystems

目前Linux支持下面12种subsystem

[cpu](https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.24; CONFIG\_CGROUP\_SCHED)  
用来限制cgroup的CPU使用率。

[cpuacct](https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.24; CONFIG\_CGROUP\_CPUACCT)  
统计cgroup的CPU的使用率。

[cpuset](https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.24; CONFIG\_CPUSETS)  
绑定cgroup到指定CPUs和NUMA节点。

[memory](https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.25; CONFIG\_MEMCG)  
统计和限制cgroup的内存的使用率，包括process memory, kernel memory, 和swap。

[devices](https://www.kernel.org/doc/Documentation/cgroup-v1/devices.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.26; CONFIG\_CGROUP\_DEVICE)  
限制cgroup创建(mknod)和访问设备的权限。

[freezer](https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.28; CONFIG\_CGROUP\_FREEZER)  
suspend和restore一个cgroup中的所有进程。

[net\_cls](https://www.kernel.org/doc/Documentation/cgroup-v1/net_cls.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.29; CONFIG\_CGROUP\_NET\_CLASSID)  
将一个cgroup中进程创建的所有网络包加上一个classid标记，用于[tc](http://man7.org/linux/man-pages/man8/tc.8.html" \t "/home/huzhengchuan/Documents\\x/_blank)和iptables。 只对发出去的网络包生效，对收到的网络包不起作用。

[blkio](https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.33; CONFIG\_BLK\_CGROUP)  
限制cgroup访问块设备的IO速度。

[perf\_event](https://www.kernel.org/doc/Documentation/perf-record.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 2.6.39; CONFIG\_CGROUP\_PERF)  
对cgroup进行性能监控

[net\_prio](https://www.kernel.org/doc/Documentation/cgroup-v1/net_prio.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 3.3; CONFIG\_CGROUP\_NET\_PRIO)  
针对每个网络接口设置cgroup的访问优先级。

[hugetlb](https://www.kernel.org/doc/Documentation/cgroup-v1/hugetlb.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 3.5; CONFIG\_CGROUP\_HUGETLB)  
限制cgroup的huge pages的使用量。

[pids](https://www.kernel.org/doc/Documentation/cgroup-v1/pids.txt" \t "/home/huzhengchuan/Documents\\x/_blank) (since Linux 4.3; CONFIG\_CGROUP\_PIDS)  
限制一个cgroup及其子孙cgroup中的总进程数。

上面这些subsystem，有些需要做资源统计，有些需要做资源控制，有些即不统计也不控制。对于cgroup树来说，有些subsystem严重依赖继承关系，有些subsystem完全用不到继承关系，而有些对继承关系没有严格要求。

Exp

cgroup相关的所有操作都是基于内核中的cgroup virtual filesystem，使用cgroup很简单，挂载这个文件系统就可以了。一般情况下都是挂载到/sys/fs/cgroup目录下，当然挂载到其它任何目录都没关系。

这里假设目录/sys/fs/cgroup已经存在，下面用到的xxx为任意字符串，取一个有意义的名字就可以了，当用mount命令查看的时候，xxx会显示在第一列

挂载一颗和所有subsystem关联的cgroup树到/sys/fs/cgroup

mount -t cgroup xxx /sys/fs/cgroup

挂载一颗和cpuset subsystem关联的cgroup树到/sys/fs/cgroup/cpuset

mkdir /sys/fs/cgroup/cpuset

mount -t cgroup -o cpuset xxx /sys/fs/cgroup/cpuset

挂载一颗与cpu和cpuacct subsystem关联的cgroup树到/sys/fs/cgroup/cpu,cpuacct

mkdir /sys/fs/cgroup/cpu,cpuacct

mount -t cgroup -o cpu,cpuacct xxx /sys/fs/cgroup/cpu,cpuacct

挂载一棵cgroup树，但不关联任何subsystem，下面就是systemd所用到的方式

mkdir /sys/fs/cgroup/systemd

mount -t cgroup -o none,name=systemd xxx /sys/fs/cgroup/systemd

在很多使用systemd的系统中，比如ubuntu 16.04，systemd已经帮我们将各个subsystem和cgroup树关联并挂载好了

dev@ubuntu:~$ mount|grep cgroup

tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)

cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release\_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)

cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)

cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)

cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)

cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)

cgroup on /sys/fs/cgroup/net\_cls,net\_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net\_cls,net\_prio)

cgroup on /sys/fs/cgroup/perf\_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf\_event)

cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)

cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)

cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)

cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)

可以通过查看/proc/[pid]/cgroup(since Linux 2.6.24)知道指定进程属于哪些cgroup。

dev@ubuntu:~$ cat /proc/777/cgroup

11:cpuset:/

10:freezer:/

9:memory:/system.slice/cron.service

8:blkio:/system.slice/cron.service

7:perf\_event:/

6:net\_cls,net\_prio:/

5:devices:/system.slice/cron.service

4:hugetlb:/

3:cpu,cpuacct:/system.slice/cron.service

2:pids:/system.slice/cron.service

1:name=systemd:/system.slice/cron.service

每一行包含用冒号隔开的三列，他们的意思分别是

cgroup树的ID， 和/proc/cgroups文件中的ID一一对应。

和cgroup树绑定的所有subsystem，多个subsystem之间用逗号隔开。这里name=systemd表示没有和任何subsystem绑定，只是给他起了个名字叫systemd。

进程在cgroup树中的路径，即进程所属的cgroup，这个路径是相对于挂载点的相对路径。

创建并挂载好一颗cgroup树之后，就有了树的根节点，也即根cgroup，这时候就可以通过创建文件夹的方式创建子cgroup，然后再往每个子cgroup中添加进程。在后续介绍具体的subsystem的时候会详细介绍如何操作cgroup。

cgroup树上创建子cgroup

dev@ubuntu:~/cgroup$ cd demo

dev@ubuntu:~/cgroup/demo$ sudo mkdir cgroup1

#删除cgroup也很简单，删除掉相应的目录就可以了

dev@ubuntu:~/cgroup/demo$ sudo rmdir cgroup1/

rmdir: failed to remove 'cgroup1/': Device or resource busy

#这里删除cgroup1失败，是因为它里面包含了子cgroup，所以不能删除，

#如果cgroup1包含有进程或者线程，也会删除失败

#先删除cgroup11，再删除cgroup1就可以了

创建新的cgroup后，就可以往里面添加进程了。注意下面几点：

在一颗cgroup树里面，一个进程必须要属于一个cgroup。

新创建的子进程将会自动加入父进程所在的cgroup。

从一个cgroup移动一个进程到另一个cgroup时，只要有目的cgroup的写入权限就可以了，系统不会检查源cgroup里的权限。

用户只能操作属于自己的进程，不能操作其他用户的进程，root账号除外。

cgroup.procs vs tasks

上面提到cgroup.procs包含的是进程ID， 而tasks里面包含的是线程ID，那么他们有什么区别呢？

#创建两个新的cgroup用于演示

dev@ubuntu:~/cgroup/demo$ sudo mkdir c1 c2

#为了便于操作，先给root账号设置一个密码，然后切换到root账号

dev@ubuntu:~/cgroup/demo$ sudo passwd root

dev@ubuntu:~/cgroup/demo$ su root

root@ubuntu:/home/dev/cgroup/demo#

#系统中找一个有多个线程的进程

root@ubuntu:/home/dev/cgroup/demo# ps -efL|grep /lib/systemd/systemd-timesyncd

systemd+ 610 1 610 0 2 01:52 ? 00:00:00 /lib/systemd/systemd-timesyncd

systemd+ 610 1 616 0 2 01:52 ? 00:00:00 /lib/systemd/systemd-timesyncd

#进程610有两个线程，分别是610和616

#将616加入c1/cgroup.procs

root@ubuntu:/home/dev/cgroup/demo# echo 616 > c1/cgroup.procs

#由于cgroup.procs存放的是进程ID，所以这里看到的是616所属的进程ID（610）

root@ubuntu:/home/dev/cgroup/demo# cat c1/cgroup.procs

610

#从tasks中的内容可以看出，虽然只往cgroup.procs中加了线程616，

#但系统已经将这个线程所属的进程的所有线程都加入到了tasks中，

#说明现在整个进程的所有线程已经处于c1中了

root@ubuntu:/home/dev/cgroup/demo# cat c1/tasks

610

616

#将616加入c2/tasks中

root@ubuntu:/home/dev/cgroup/demo# echo 616 > c2/tasks

#这时我们看到虽然在c1/cgroup.procs和c2/cgroup.procs里面都有610，

#但c1/tasks和c2/tasks中包含了不同的线程，说明这个进程的两个线程分别属于不同的cgroup

root@ubuntu:/home/dev/cgroup/demo# cat c1/cgroup.procs

610

root@ubuntu:/home/dev/cgroup/demo# cat c1/tasks

610

root@ubuntu:/home/dev/cgroup/demo# cat c2/cgroup.procs

610

root@ubuntu:/home/dev/cgroup/demo# cat c2/tasks

616

#通过tasks，我们可以实现线程级别的管理，但通常情况下不会这么用，

#并且在cgroup V2以后，将不再支持该功能，只能以进程为单位来配置cgroup

#清理

root@ubuntu:/home/dev/cgroup/demo# echo 610 > ./cgroup.procs

root@ubuntu:/home/dev/cgroup/demo# rmdir c1

root@ubuntu:/home/dev/cgroup/demo# rmdir c2

root@ubuntu:/home/dev/cgroup/demo# exit

exit

Question

Link From:

http://www.tuicool.com/articles/zmuUZjI